Velocity 中文文档

wizardforcel

Published with GitBook



目錄

Velocity 中文文档

1. 关于

Velocity 用户指南旨在帮助页面设计者和内容提供者了解Velocity 和其简单而又强大的脚本语言（Velocity Template Language (VTL)）。本指南中有很多示例展示了用Velocity来讲动态内容嵌入到网站之中，但是所有的VTL examples 都同演示用于所有的页面和模版。

感谢选择Velocity!

2. 什么是Velocity?

Velocity 是一个基于Java的模版引擎。它允许web 页面设计者引用JAVA代码预定义的方法。Web 设计者可以根据MVC模式和JAVA程序员并行工作，这意味着Web设计者可以单独专注于设计良好的站点，而程序员则可单独专注于编写底层代码。Velocity 将Java 代码从web页面中分离出来，使站点在长时间运行后仍然具有很好的可维护性，并提供了一个除JSP和PHP之外的可行的被选方案。

Velocity可用来从模板产生web 页面，SQL, PostScript以及其他输出。他也可用于一个独立的程序以产生源代码和报告，或者作为其它系统的一个集成组件。这个项目完成后，Velocity将为Turbine web 应用程序框架提供模板服务。Velocity+Turbine 方案提供的模板服务将允许web 应用按真正的mvc模式进行开发。

3. Velocity 可以做什么?

3.1. Mud Store 示例

假设你是一个专门销售泥浆（MUD）的在线商店的页面设计者。我们称他为"The Online Mud Store"。生意很好。客户订购各种各样的类型和数量的泥浆。他们使用他们的用户名和密码登陆到商店中来，就可以浏览他们的订货和购买其他东西。现在，赤土陶泥正在促销，这是一种很常用的泥巴。一少部分顾客很有规律的购买一种亮红土Bright Red Mud，这也是促销产品，但是不太常用，因此被移到页面的边缘。所有顾客的信息都在数据库中被跟踪，因此有一天问题出现了：为什么不使用Velocity来定位目标客户，这些客户对某种类型的产品特别感兴趣？

Velocity 使针对访问者个性的WEB页面客户化（个性化）非常容易。作为一个在线泥巴商店的站点设计者，以想在客户以登陆进展点后就看到它们想看的页面。

你遇到你公司的软件工程师，每个人都认为$customer 将保持当前登陆进入的客户信息，而$mudsOnSpecial 将士当前所有促销的泥巴。$flogger 对象包含有助于促销的方法。对于当前的任务，让我们仅关注这三个问题。记住，你不需要担心软件工程师如何从数据库中取得顾客信息，但你必须知道他们可以。这样可以使你专注于你的工作而软件工程师则忙于他们自己的工作。

你可以在你的页面中嵌入如下的VTL语句：

```
<HTML>
<BODY>
Hello $customer.Name!
<table>
#foreach( $mud in $mudsOnSpecial )
   #if ( $customer.hasPurchased($mud) )
      <tr>
        <td>
          $flogger.getPromo( $mud )
        </td>
      </tr>
   #end
#end
</table>
```

foreach语句的细节将进一步细说，但重要的是这个短小的脚本居然可以在你的站点上运行。当有一个倾向于亮红土的顾客登陆进来时，亮红土正在促销，这就是这个顾客所看到的，并且促销显示非常显著。如果另外一个长期购买赤陶土的顾客登陆进来，赤陶土促销的提示信息则应该在前面中间位置。Velocity是非常灵活的，受限的只是你的创造力。

写在VTL参考文档中的是其他Velocity 元素，他们一起给你很强大的能力和灵活性以创建很好的站点。待你更加了解这些元素，就可以开始释放Velocity的强大动力。

4. Velocity模板语言(VTL): 介绍

Velocity模板语言(VTL)旨在为Web页面结合动态内容提供最容易、简单和简洁的方法。即使有一点或者没有编程经验的页面设计者也可以很快能为页面提供动态内容。

VTL 使用引用（references）来将动态内容嵌入web页面，每个变量就是某一个类型的引用。变量实际上是一个可以调用定义在java代码中的内容的引用，或者它可以从页面内的VTL语句得出自身的值。下面是一个例子，说明可以嵌入到HTML文档中的VTL语句。

```
#set( $a = "Velocity" )
```

这个VTL语句，就像所有的VTL语句一样，以#字符开始，并跟着一个指令set。当一个在线访问这请求页面时， Velocity 模伴引擎在页面内搜索所有#字符，然后决定是哪一个标记了VTL语句的开始，哪个标记不需要VTL做什么动作。

#字符后面紧跟一个指令set.。set指令使用一个括在括号内的表达式---一个等式将一个值指派给一个变量。变量在等号的左边而值在等号的右边。

在上面的示例中，变量是$a值是Velocity。这个变量就象其他引用一样，以一个$字符开始。值通常在引号之中，对Velocity来说一般没有类型冲突的问题，因为只有字符串 (基于文本的信息)可以传递给变量。

下面的主要规则可能有助于理解Velocity 是如何工作的：引用以$开头用于取得什么东西，而指令以#开始用于做什么事情。

在上面的例子中，#set用于将一个值指派给一个变量。而变量$a则可以用来在模板中输出"Velocity" 。

5. Hello Velocity World!

一旦一个值被赋给一个变量，便可以在HTML中随处引用它。在下面的示例中，先给变量$foo赋值然后引用它。

```
<html>
<body>
#set( $foo = "Velocity" )
Hello $foo World!
</body>
<html>
```

这个页面的结果是输出"Hello Velocity World!"。

为了使包含VTL 指令的语句具有可读性，我们鼓励每个VTL语句在一个新行开始，虽然并不一定要这样做。set将随后深入解释。

6. 注释

可以用注释加入描述性文本，他们并不在模板引擎中输出。注释可以有助于你的记忆或者想其他人解释你的VTL语句正在做什么。

```
## This is a single line comment.
```

单行注释以##开始，并在本行结束。如果需要加入多行注释，并不需要加入很多的单行注释。多行注释，以#*开始并以*#结束可以处理这种情况。

```
This is text that is outside the multi-line comment.
Online visitors can see it.

#*
 Thus begins a multi-line comment. Online visitors won't
 see this text because the Velocity Templating Engine will
 ignore it.
*#

Here is text outside the multi-line comment; it is visible.
```

下面事一些例子说明单行注释和多行注释如何工作。

```
This text is visible. ## This text is not.
This text is visible.
This text is visible. #* This text, as part of a multi-line comment,
is not visible. This text is not visible; it is also part of the
multi-line comment. This text still not visible. *# This text is outside
the comment, so it is visible.
## This text is not visible.
```

还有第三种注释， VTL 注释块，可以用来存储诸如文档作者、版本信息等。

```
#**
This is a VTL comment block and
may be used to store such information
as the document author and versioning
information:
@author
@version 5
*#
```

7. 引用

VTL中有三种类型的引用：变量，属性和方法。作为使用VTL的设计者，你和你的工程师必须在饮用的特定命名上取得一致，以便在你的模板中正确的使用他们。

有关引用的所有参数都处理为字符串对象。Everything coming to and from a reference is treated as a String object. 假如有一个对象表示$foo (比如说是整型对象)，Velocity 将调用其 toString() 方法来将此对象转换为一个字符串。

7.1. 变量Variables

变量的简略标记是有一个前导"$"字符后跟一个 VTL 标识符（Identifier.）组成。一个VTL 标识符必须以一个字母开始(a .. z或 A .. Z)。剩下的字符将由以下类型的字符组成：

- 字母 (a .. z, A .. Z)
- 数字 (0 .. 9)
- 连字符("-")
- 下划线 ("_")

下面是一些有效的变量引用:

```
$foo
$mudSlinger
$mud-slinger
$mud_slinger
$mudSlinger1
```

当VTL 引用一个变量时，比如$foo，变量可以从模板的set指令取得值，也可以从

Java 代码中取得。例如，如果Java 变量$foo在模板被请求的时候具有值bar，则bar将替换页面中的所有$foo的实例。或者，如果包含下面的语句：

```
#set( $foo = "bar" )
```

紧跟指令后的所有$foo的实例的输出将会一样值。

7.2. 属性

VTL引用的第二种元素是属性，而属性具有独特的格式。属性的简略标记识前导符$后跟一个VTL 标识符，在后跟一个点号(".")最后又是一个VTL 标识符。这是一些有效的示例：

```
$customer.Address
$purchase.Total
```

请看第一个例子，$customer.Address.。他有两种意思。它可以意味着，查询由customer标是的哈希表并按关键字Address返回值。但是$customer.Address也可能引用一个方法（下述，$customer.Address可能是$customer.getAddress().的缩写。当一个页面被请求时，Velocity 将决定这两种可能到底是哪一个，然后返回相应的值。

7.3. 方法

方法在JAVA代码中定义，并作一些有用的事情，比如运行一个计算器或者作出一个决定。方法是实际上也是引用，由前导符"$"后跟一个VTL 标识符，后跟一个VTL 方法体（Method Body）。 VTL 方法体由一个VTL 标识符后跟一个左括号，再跟可选的参数列表，最后是右括号。下面是一些有效的方法示例：

```
$customer.getAddress()
$purchase.getTotal()
$page.setTitle( "My Home Page" )
$person.setAttributes( ["Strange", "Weird", "Excited"] )
```

前面两个例子-- $customer.getAddress()和$purchase.getTotal() – 看起来有点象上面属性一节中所用的样子，$customer.Address和$purchase.Total.。如果你想这些例子在某些方面相关，那你就对了。

VTL 属性可以为VTL方法用作简略标记。属性$customer.Address具有和方法$customer.getAddress() 完全一样的效果。属性和方法的主要不同点是方法中可以添加参数列表。

简略标记可以用在下面的方法中：

```
sun.getPlanets()
$annelid.getDirt()
$album.getPhoto()
```

我们或许希望方法可以为我们放回属于太阳系的行星的名字，喂养我们的蚯蚓，或者从相册中返回一张照片。下面只有长的那个标记是可以工作的方法：

```
$sun.getPlanet( ["Earth", "Mars", "Neptune"] )
## 不能将参数列表传递给$sun.Planets

$sisyphus.pushRock()
## Velocity 假定我意思是$sisyphus.getRock()

$book.setTitle( "Homage to Catalonia" )
## 不能传递一个参数列表
```

8. 形式引用符Formal Reference Notation

引用的简略符号如上所述，但是另外还有一种引用的形式符号，示例如下：

```
${mudSlinger}
${customer.Address}
${purchase.getTotal()}
```

在大多数情况下，我们将使用引用的简略符号，但在一些情况下，也需要拥戴哦形式引用符以便正确处理。

假定你正在纸片上构件一个句子，将使用$vice作为句子中名词的词根。我们的目标是允许人们选择词根，然后产生以下两种结果之一：

```
"Jack is a pyromaniac."
```

或者

```
"Jack is a kleptomaniac."。
```

在这种情况下，使用简略符号是不太充分的。考虑到下面的例子：

```
Jack is a $vicemaniac.
```

这里有个不确定性， Velocity 假定 $vicemaniac，（而不是 $vice）是一个你想要使用的标识符。找不到$vicemaniac的值，他将返回$vicemaniac。使用形式符号便可解决这个问题：

```
Jack is a ${vice}maniac
```

现在Velocity 知道$vice（而不是$vicemaniac）是一个引用。形式符号常用在饮用咋模板中和文本直接邻近的地方。

9. 安静引用符Quiet Reference Notation

当 Velocity 遇到一个位定义的引用时，其通常行为是输出这个引用的映像。比如，假设下面的引用出现在模板中的一部分：

```
<input type="text" name="email" value="$email"/>
```

当表单初次装入时，变量引用$email无值，你宁愿是一个空白域而不是具有值"$email"。使用安静引用符可以绕过Velocity的常规行为，在VTL中不用$email而是用$!email 符号。所以，上面的例子将会看起来像下面的样子：

```
<input type="text" name="email" value="$!email"/>
```

现在，当表单初次装入时， $email 仍然没有值，但是将输出空字符串而不是"$email"。

形式和安静引用符可以一起使用，如下所示：

```
<input type="text" name="email" value="$!{email}"/>
```

11. Case Substitution

现在你大致了解了引用，可以在模板中使用它们了。Velocity 采用了很多JAVA原理的优点，模板设计人员会发现非常容易使用。例如：

```
$foo

$foo.getBar()
## is the same as
$foo.Bar

$data.getUser("jon")
## is the same as
$data.User("jon")

$data.getRequest().getServerName()
## is the same as
$data.Request.ServerName
## is the same as
${data.Request.ServerName}
```

这个例子显示了引用的一些其他用法。Velocity 借鉴了Java的自省和组件bean特征，来解决引用名在上下文中作为对象和对象方法的问题。可以在你的模板的任何地方插入引用和求值。

Velocity, 建模在Sun Microsystems定义的BEAN规范之上，是大小写敏感的；开发者努力捕捉和纠正可能出现的用户错误。当方法getFoo() 在模板中通过$bar.foo引用时，Velocity 首先尝试$getfoo。如果失败，他会再尝试 $getFoo。类似地，当一个模板引用到 $bar.Foo，Velocity 将尝试 $getFoo() 先，然后尝试 getfoo()。

注意：模板中引用示例变量的问题仍然没有解决。只有引用等价于JavaBean的 getter/setter 方法解决了。(比如 $foo.Name 解决了到类 Foo的 getName() 示例方法的引用，但不能引用Foo的一个公共实例变量Name)。

12. 指令

因为指令（使用脚本来有效操控JAVA代码的输出）允许页面设计员真正专注于咱点的外观和内容设计，引用允许模板设计员为Web页面产生动态内容。

12.1. #set

`#set` 指令用来为引用设置相应的值。值可以被值派给变量引用或者是属性引用，而且赋值要在括号里括起来。

```
#set( $primate = "monkey" )
#set( $customer.Behavior = $primate )
```

赋值的左边必须是一个变量应用或者是属性引用。右边可以是下面的类型之一：

- 变量引用
- 字面字符串
- 属性引用
- 方法引用
- 字面数字
- 数组列表

这些例子演示了上述的每种类型：

```
#set( $monkey = $bill ) ## variable reference
#set( $monkey.Friend = "monica" ) ## string literal
#set( $monkey.Blame = $whitehouse.Leak ) ## property reference
#set( $monkey.Plan = $spindoctor.weave($web) ) ## method reference
#set( $monkey.Number = 123 ) ##number literal
#set( $monkey.Say = ["Not", $my, "fault"] ) ## ArrayList
```

注意：最后一个例子中，在方括号[..] 中定义的项目可以被ArrayList 类定义的方法访问。比如，你可以使用$monkey.Say.get(0)访问上述的第一个元素。

右边也可以是一个简单的算术表达式:

```
#set( $value = $foo + 1 )
#set( $value = $bar - 1 )
#set( $value = $foo * $bar )
#set( $value = $foo / $bar )
```

如果右边是一个属性或方法引用，取值是NULL，他将不会赋值给左边。通过这种机制将一个存在的引用从上下文中删除是不可能的。这对Velocity的新手可能会混淆。例如：

```
#set( $result = $query.criteria("name") )
The result of the first query is $result

#set( $result = $query.criteria("address") )
The result of the second query is $result
```

如果， $query.criteria("name") 放回字符串"bill"，而$query.criteria("address") 返回 null，上述VTL 将解释为：

```
The result of the first query is bill

The result of the second query is bill
```

这往往会给那些想构建#foreach循环来试图通过属性和方法引用来设置一个引用的新手带来困惑，下面马上通过#if指令测试一下。例如：

```
#set( $criteria = ["name", "address"] )

#foreach( $criterion in $criteria )

    #set( $result = $query.criteria($criterion) )

    #if( $result )
        Query was successful
    #end

#end
```

在上面的例子中，依靠$result 的去值来决定查询是否成功恐怕不是英明的做法。当$result 被#set设置后(添加到上下文中)，他就不能再被设值为null (从上下文中删除)。

我们对此的解决方法是预设$result 为 false。然后如果 $query.criteria() 调用失败，你就可以检查之。

```
#set( $criteria = ["name", "address"] )

#foreach( $criterion in $criteria )

    #set( $result = false )
    #set( $result = $query.criteria($criterion) )

    #if( $result )
        Query was successful
    #end

#end
```

不象其他Velocity 指令， #set 指令没有#end 语句。

12.2. 字面字符串

当使用#set 指令时，括在双引号中的字面字符串将解析和重新解释，如下所示：

```
#set( $directoryRoot = "www" )
#set( $templateName = "index.vm" )
#set( $template = "$directoryRoot/$templateName" )
$template
```

输出将会是：

```
www/index.vm
```

然而，当字面字符串括在单引号中时，他将不被解析：

```
#set( $foo = "bar" )
$foo
#set( $blargh = '$foo' )
$blargh
```

输出是：

```
Bar
 $foo
```

默认情况下，使用单引号来渲染未解析文本在Velocity是有效的。这种特征可以通过编辑velocity.properties 中的stringliterals.interpolate=false来改变。

12.3. 条件

12.3.1 If / ElseIf / Else

Velocity中的#if 指令允许在页面生成时，在IF条件为真的情况下包含文本。例如：

```
#if( $foo )
   <strong>Velocity!</strong>
#end
```

变量 $foo 先求值，以决定是否为真。在这两种情况下为真： (i) $foo 是一个逻辑变量并具有真的值，或者 (ii) 值非空。要记住Velocity 上下文仅包括对象，所以当我们说"布尔"'boolean'时，他会被表示为"布尔类"(Boolean class)。这对即使是返回布尔类型的方法也是真的—自省架构将返回一个具有相同逻辑值的布尔类。

如果求值为真时， #if 和 #end 语句之间的内容将输出。在这种情况下，如果 $foo 为真，输出将是"Velocity!"。相反，如果 $foo 具有一个null 值，或者逻辑假，语句求值为假，则没有输出。

一个 #elseif 或者 #else 项可以用在#if 语句中。请注意， Velocity 模板引擎将在第一个为真的表达式时停止。下面的例子中，假设$foo 具有值15 而 $bar 等于 6。

```
#if( $foo < 10 )
    <strong>Go North</strong>
#elseif( $foo == 10 )
    <strong>Go East</strong>
#elseif( $bar == 6 )
    <strong>Go South</strong>
#else
    <strong>Go West</strong>
#end
```

在这个例子中，$foo 大于10，所以前面两个比较失败。接下来比较$bar 和6，结果为真，所以输出为Go South。

请注意在现在， Velocity的数值比较约束为整型—其他类型都将求值为false。仅有一个例外是等于'==',这时Velocity 要求等号两边的对象具有相同的类型。

12.3.2 关系和逻辑操作符

Velocity 使用等式操作符来决定两个变量间的关系。这里是一个简单的例子演示如何使用等式操作符：

```
#set ($foo = "deoxyribonucleic acid")
#set ($bar = "ribonucleic acid")

#if ($foo == $bar)
 In this case it's clear they aren't equivalent. So...
#else
 They are not equivalent and this will be the output.
#end
```

Velocity 也具有逻辑AND, or 和 NOT 操作符。更进一步的信息，请看VTL参考手册VTL Reference Guide 。下面是一些演示如何使用逻辑操作符的例子：

```
## logical AND

#if( $foo && $bar )
   <strong> This AND that</strong>
#end
```

例子中#if() 指令仅在$foo 和$bar 斗为真的时候才为真。如果$foo 为假，则表达式也为假；并且 $bar 将不被求值。如果 $foo 为真，Velocity 模板引擎将继续检查$bar;的值，如果 $bar 为真，则整个表达式为真。并且输出This AND that 。如果 $bar 为假，将没有输出因为整个表达式为假。

逻辑OR 的工作方式相同，唯一的例外是其中一个表达式要被求值，以便决定整个表达式是否为真。请看下面的例子：

```
## logical or

#if( $foo || $bar )
    <strong>This or That</strong>
#end
```

如果 $foo 为真，Velocity 模板引擎就不需要去察看$bar 的值，不管 $bar 是否为真，真个表达式都为真，因此输出This or That 。如果 $foo 为假，$bar 就必须检查其值了。在这种情况下，如果$bar 也是为假，表达式将为假，没有任何输出。当然，如果$bar 为真，则真个表达式为真，输出This or That。

对于逻辑NOT 操作符，只有一个操作数：

```
##logical NOT

#if( !$foo )
 <strong>NOT that</strong>
#end
```

这里，如果$foo 为真，!$foo 求值为假，没有输出。如果$foo 为假，!$foo 求值为真，输出 NOT that 。请当心，不要和安静引用quiet reference $!foo 混淆它们是完全不同的。

12.4. 循环

12.4.1. Foreach 循环

#foreach 元素允许进行循环，例如：

```
<ul>
#foreach( $product in $allProducts )
    <li>$product</li>
#end
</ul>
```

这个#foreach 循环将导致$allProducts 列表 (对象) 为查询所有的产品$products (目标)遍历一遍。每次经过循环，从$allProducts取得的值将置于$product 变量之中。

$allProducts 变量的内容是一个矢量，一个哈希表或者数组。赋给$product 变量的值是一个Java 对象并且可以从一个类似的变量引用。例如，如果 $product 真是一个Java的产品类，其名称可以通过引用$product.Name 方法来检索(即: $Product.getName())。

我们假定 $allProducts 是一个哈希表。如果你想检索关键字的值或者在哈希表中的对象，你可以使用以下的代码：

```
<ul>
#foreach( $key in $allProducts.keySet() )
    <li>Key: $key -> Value: $allProducts.get($key)</li>
#end
</ul>
```

Velocity 提供一个更容易的方式或的循环计数，以便你可以做下面类似的工作：

```
<table>
#foreach( $customer in $customerList )
    <tr><td>$velocityCount</td><td>$customer.Name</td></tr>
#end
</table>
```

循环计数变量的缺省名称是$velocityCount，在velocity.properties 配置文件中标明。默认情况下，该变量从1开始计数，但是可以在velocity.properties 文件中设为从0或者1开始。下面是velocity.properties 文件中循环变量设置一节:

```
# Default name of the loop counter
# variable reference.
directive.foreach.counter.name = velocityCount

# Default starting value of the loop
# counter variable reference.
directive.foreach.counter.initial.value = 1
```

12.5. 包含

#include 脚本元素允许模板设计人员包含（导入）本地文件，这个文件将插入到#include 指令被定义的地方。文件的内容并不通过模板引擎来渲染。处于安全的原因，被包含的文件只可以放在TEMPLATE_ROOT下。

```
#include( "one.txt" )
```

#include 指令引用的文件在双引号内。如果超过一个文件，其间用逗号隔开。

```
#include( "one.gif","two.txt","three.htm" )
```

被包含的文件并不是一定要用文件名来引用，事实上，最好的办法是使用变量而不是文件名。这在根据规则决定何时提交页面时，决定目标输出是很有用的。

```
#include( "greetings.txt", $seasonalstock )
```

12.6. 解析

#parse 脚本元素允许页面设计员导入包含VTL的本地文件。 Velocity将解析和渲染指定的模板。

```
#parse( "me.vm" )
```

就象 #include 指令，#parse 可以使用变量而不是一个实在的模板文件。#parse 引用的模板文件必须包含的TEMPLATE_ROOT指定的目录之下。和 #include 指令不一样， #parse 只有一个参数。

VTL 模板templates can have #parse statements referring to templates that in turn have #parse statements. By default set to 10, the parse_directive.maxdepth line of the velocity.properties allows users to customize maximum number of #parse referrals that can occur from a single template. (Note: If the parse_directive.maxdepth property is absent from the velocity.properties file, Velocity will set this default to 10.) Recursion is permitted, for example, if the template dofoo.vm contains the following lines:

```
Count down.
#set( $count = 8 )
#parse( "parsefoo.vm" )
All done with dofoo.vm!
```

It would reference the template parsefoo.vm, which might contain the following VTL:

```
$count
#set( $count = $count - 1 )
#if( $count > 0 )
    #parse( "parsefoo.vm" )
#else
    All done with parsefoo.vm!
#end
```

After "Count down." is displayed, Velocity passes through parsefoo.vm, counting down from 8. When the count reaches 0, it will display the "All done with parsefoo.vm!" message. At this point, Velocity will return to dofoo.vm and output the "All done with dofoo.vm!" message.

12.7. 停止

#stop 脚本允许模板设计员停止模板引擎的执行，并返回。这通常用作调试。

```
#stop
```

12.10. 宏

#macro 脚本元素允许模板设计者在VTL 模板中定义重复的段。 Velocimacros 不管是在复杂还是简单的场合都非常有用。下面这个Velocimacro，仅用来节省击键和减少排版错误，介绍了一些Velocity宏的概念。

```
#macro( d )
<tr><td></td></tr>
#end
```

在例子中，Velocimacro定义为d，它可以象调用其他VTL指令一样的形式来进行调用：

```
#d()
```

当这个模板被调用时， Velocity 将 #d() 替换为一个单行的空表格。

Velocimacro 可以带一些参数，也可以不带参数（如上例所示）。但在他被调用时，所带的参数必须和其定义时的参数一样。很多Velocimacros 定义为不止一个参数。下面这个宏带有两个参数，一个颜色，一个数组。

```
#macro( tablerows $color $somelist )
#foreach( $something in $somelist )
    <tr><td bgcolor=$color>$something</td></tr>
#end
#end
```

在这个例子中定义的Velocimacro，名为tablerows, 要求两个参数。第一个参数代替$color, 第二个代替$somelist。

可以写进VTL 模板中的东西都可以写进Velocimacro 的主体部分。tablerows 宏其实是一个foreach 语句。在#tablerows 宏的定义中有两个#ende语句，第一个属于#foreach, 第二个结束宏定义。

```
#set( $greatlakes = ["Superior","Michigan","Huron","Erie","Ontario"] )
#set( $color = "blue" )
<table>
    #tablerows( $color $greatlakes )
</table>
```

请注意$greatlakes 替换了$somelist。这样，当#tablerows 宏被调用时，将产生以下输出：

```
<table>
    <tr><td bgcolor="blue">Superior</td></tr>
    <tr><td bgcolor="blue">Michigan</td></tr>
    <tr><td bgcolor="blue">Huron</td></tr>
    <tr><td bgcolor="blue">Erie</td></tr>
    <tr><td bgcolor="blue">Ontario</td></tr>
</table>
```

Velocimacros 在Velocity 模板语句内定义，这意味着它在同一站点内的其他Velocity 模板中并不有效。定义一个宏，并使其与其他模板共享很具有明显的优点：他减少了在大量的模板内重复定义宏的工作，并减少了出错的机会，并确保对其他宏的改变对其他所有模板有效。

但如果 #tablerows($color $list) 宏是在一个Velocimacros 模板库内定义的，它就可以被其他常规模板所用。当然，它可以用于各种目的，也可重用多次。在表示所有真菌类（fungi）的 mushroom.vm 模板中，#tablerows 宏可以被用来列出典型的蘑菇。

```
#set( $parts = ["volva","stipe","annulus","gills","pileus"] )
#set( $cellbgcol = "#CC00FF" )
<table>
#tablerows( $cellbgcol $parts )
</table>
```

我们对mushroom.vm执行请求，Velocity 将在模板库内找到#tablerows 宏 (在 velocity.properties 文件中定义)并产生以下输出：

```
<table>
    <tr><td bgcolor="#CC00FF">volva</td></tr>
    <tr><td bgcolor="#CC00FF">stipe</td></tr>
    <tr><td bgcolor="#CC00FF">annulus</td></tr>
    <tr><td bgcolor="#CC00FF">gills</td></tr>
    <tr><td bgcolor="#CC00FF">pileus</td></tr>
</table>
```

12.10.1. Velocimacro 参数

Velocimacros 的参数可以是以下的VTL元素：

- 引用（Reference）: 以 '$' 打头的元素
- 字面字符串（String literal）：比如"$foo" 或 'hello'
- 字面数字: 1, 2 ....
- 整数范围: [1..2] 或 [$foo .. $bar]
- 对象数组: ["a", "b", "c"]
- 布尔真
- 布尔假

当把引用作为参数传递给Velocimacros时，请注意引用是按“名字”传递的。这意味着他们的值在每次使用他们的Velocimacro中产生。这个特性允许你在方法调用是传递引用，并在每次使用时进行方法调用。例如，Fo，当调用下面的Velocimacro 时，

```
#macro( callme $a )
         $a $a $a
     #end

     #callme( $foo.bar() )
```

结果是，在方法bar() 中，引用 $foo 被调用了3次。

咋看时，这个特征让人吃惊，当当你考虑一下Velocimacros的原本动机 – 在VTL模板中避免很多“剪切复制”操作—你就会明白。它允许你将无状态对象，比如在一个颜色表格行内重复产生一些颜色次序的对象，传递给Velocimacro。

如果你需要使用这个特征，你通常可以从方法内取得一个值，作为一个新的引用传递给宏：

```
 #set( $myval = $foo.bar() )
     #callme( $myval )
```

12.10.2. Velocimacro 属性

在velocity.properties 文件中有数行定义可以用来灵活实现Velocimacros。详细情况请参见开发指南（Developer Guide）。

velocimacro.library –是一个逗号分隔的所有Velocimacro 模板库的列表。默认情况下，Velocity 搜寻一个单一的库VM_global_library.vm.。预先配置的模板路径用来查找Velocimacro 库。

velocimacro.permissions.allow.inline –这个属性决定Velocimacros 是否可以在常规模板内定义，取值为逻辑True或者False。默认情况下，设置为true，允许设计者在产规模板内定义宏。

velocimacro.permissions.allow.inline.to.replace.global –逻辑true 或者false，允许标明是否允许在常规模板内定义的Velocimacro代替在模板库中定义并通过velocimacro.library属性在启动时装入的全局宏。默认设置为false。

velocimacro.permissions.allow.inline.local.scope –逻辑true 或者false，默认值为false。控制是否在模板内定义的Velocimacros仅在定义它的模板内可见。换句话说，如果设置为true，一个模板可以定义仅能被他所用的宏。你可以用它来做一些漂亮的宏，如果一个全局调用另一个全局宏，在局部（inline）范围内，当被一个模板调用时，该模板可以定义一个被第一个全局宏调用的第二个全局宏的私有实现。其他所有模板都不受影响。

velocimacro.context.localscope –逻辑值true 或者 false，缺省值为false。但设置为true时，所有在Velocimacro 内通过 #set() 进行的修改都将被视为Velocimacro 的本地行为，不会影响到其上下文。

velocimacro.library.autoreload –此属性控制Velocimacro 库的自动载入。缺省值为false。如果设置为true，被调用的Velocimacro得源库将被检查是否改变，并在必要是重新载入。这将使你可以改变和测试Velocimacro 库，而不必重新启动应用服务器或者servlet容器，就象你工作在常规模板一样。这个模时仅在资源载入器的缓存模时被关闭的情况下有效 (如 file.resource.loader.cache = false)。此特征为开发时设计，不要在生产模式时使用。

12.10.3. Velocimacro Trivia

当前， Velocimacros 在其首次在模版中使用前必须首先定义它。这意味着， #macro() 宣称应该在使用Velocimacros之前。

如果你想#parse() 一个包含#macro() 指令的模板，记住这个非常重要。因为#parse() 在运行时发生，解析器在解析时要决定是否模版中一个看起来像VM的元素真是VM，所以解析一系列VM 宣称可能并不能如愿地工作的很好。为避免如此，可以简单地使用velocimacro.library的办法，使Velocity 在启动时载入VM。

13. Getting literal

Velocity使用特殊字符$和#来帮助它工作，所以如果要在template里使用这些特殊字符要格外小心。本节将讨论$字符。

13.1. 货币字符

这是没有问题的："I bought a 4 lb. sack of potatoes at the farmer's market for only $2.50!"，VTL中使用$2.5这样的货币标识是没有问题得的，VTL不会将它错认为是一个reference，因为VTL中的reference总是以一个大写或者小写的字母开始。

13.2. 转义 有效的 VTL 指令

某些情况使用Velocity可能会觉得很烦恼。逃避特殊符是处理出现在你的模板中VTL特殊符有效方法，就是使用反斜杠(“\”)。

```
#set( $email = "foo" )
$email
```

假如Velocity在你的模板中遇到$email，它会搜索上下文，得到相应的值。这里的输出是foo，因为$email被定义了。假如$email没有被定义，输出会是$email。

设想$email被定义了(例如，它的值是foo)，而且你想输出$email。这里有几种方法能达到目的，但是最简单的是使用逃避符。

```
## The following line defines $email in this template:
#set( $email = "foo" )
$email
\$email
```

将显示为：

```
foo
$email
```

注意到“\”屏蔽了左边的“$”。屏蔽左边规则，使得\$email显示为\$email。那些例子与$email没有定义相比较。

```
$email
\$email
\\$email
\\\$email
```

将显示为：

```
$email
\$email
\\$email
\\\$email

\\$email\\\$email
```

注意Velocity处理定义了的references与没有定义的不一样。这里set$foo的值为gibbous。

```
#set( $foo = "gibbous" )
$moon = $foo
```

输出会是： `$moon=gibbous,$moon` 按照字面上输出因为它没有定义，gibbous替代$foo输出。避开VTL的directives还有其他方法，在Directives那章节会更详细描述。

13.3. 转义 无效的 VTL 指令

VTL 可以通过反斜杠("/")来进行转义，directives can be escaped with the backslash character in a manner similar to valid VTL references.

```
## #include( "a.txt" ) renders as <contents of a.txt>
#include( "a.txt" )

## /#include( "a.txt" ) renders as /#include( "a.txt" )
/#include( "a.txt" )

## //#include ( "a.txt" ) renders as /<contents of a.txt>
//#include ( "a.txt" )
```

在转义在一个单一指令内包含多个脚本元素（比如f-else-end语句）的指令时应多加小心。下面是一个典型的VTL if语句；

```
#if( $jazz )
    Vyacheslav Ganelin
#end
```

如果 $jazz为 true，输出是

```
Vyacheslav Ganelin
```

如果 $jazz 为false，将没有输出。转义脚本元素将改变输出。考虑下面的情况；

```
/#if( $jazz )
    Vyacheslav Ganelin
/#end
```

不管 $jazz 是真或假，输出都是

```
#if($ jazz )
     Vyacheslav Ganelin
 #end
```

事实上，因为所有脚本元素都被转义了， $jazz 永远不会被求值。将设反斜杠在被合法转义的脚本元素之前

```
//#if( $jazz )
   Vyacheslav Ganelin
//#end
```

这时，如果$jazz 为真，输出是

```
/ Vyacheslav Ganelin
/
```

为理解这个情况，请注意在一个新行结束是将在输出中忽略新的一行。因此，经过#if()前的'//'加工后，#if()块紧跟第一个'/'。最后一个/位于新的一行，因为在'Ganelin'后又一个新行，所以，最后的那个位于#end 之前的//是语句块的一部分。

如果 $jazz 为false，这里将没有输出。注意，在开始破坏了if语句的情况将不能被正确转义：

```
///#if( $jazz )
    Vyacheslave Ganelin
//#end
```

这里，#if 被转义，但有一个#end 被保留了；所以有多个结束语句将导致解析错误。

14. VTL 格式化问题

虽然在本指南中的VTL经常显示在新行中或者有空格，但是下面的VTL

```
#set( $imperial = ["Munetaka","Koreyasu","Hisakira","Morikune"] )
#foreach( $shogun in $imperial )
    $shogun
#end
```

和下面的写法同样有效。

```
Send me #set($foo = ["$10 and ","a cake"])#foreach($a in $foo)$a #end please.
```

Velocity的行为并不受空格的影响，前述的指令也可以写成：

```
Send me
#set( $foo = ["$10 and ","a cake"] )
#foreach( $a in $foo )
$a
#end
please.
```

或者

```
Send me
#set($foo       = ["$10 and ","a cake"])
                 #foreach           ($a in $foo )$a
         #end please.
```

上面每种写法结果都一样。

15. 其它特征和杂项

15.1. 数学特征

Velocity 有一些内建的数学功能，可以使用set指令用在模版中。下面的共识分别演示了加减乘除运算：

```
#set( $foo = $bar + 3 )
#set( $foo = $bar - 4 )
#set( $foo = $bar * 6 )
#set( $foo = $bar / 2 )
```

当进行除法运算时，结果将会是整数。When a division operation is performed, the result will be an integer. 余数则可以通过模(%)运算获得。

```
#set( $foo = $bar % 5 )
```

在Velocity 中，只有整数可以进行数学运算；如果执行非整数的数学运算，将被记录下来，并返回null 。

15.2. 范围操作符

范围操作符可以和#set 和#foreach 语句一起使用。有助于产生一个整数的目标数组，范围操作符有以下的结构：

```
[n..m]
```

n 和 m 都必须是整数或者可以产生整数。不管 m 大于或者小于n 都没关系；在m小于n这种情况下，范围可以向下计数。下面是使用范围操作符的例子：

第一个例子

```
#foreach( $foo in [1..5] )
$foo
#end
```

第二个例子

```
#foreach( $bar in [2..-2] )
$bar
#end
```

第三个例子

```
#set( $arr = [0..1] )
#foreach( $i in $arr )
$i
#end
```

第四个例子

```
[1..3]
```

他们分别产生一下输出

```
1 2 3 4 5

2 1 0 -1 -2

0 1

 [1..3]
```

范围操作符和#set 和#foreach 指令一起使用时，只是产生数组。

页面设计人员在设计具有相同尺寸的表格时，有时没有足够的数据来填充，他们会发现范围操作符非常有用。

15.3. 进阶：转义和!

当一个引用被! 字符处于静寂模式，并且! 字符在转义符/ 前出现，应用将用一种特别的方式处理。请注意他和常规转义的不同，下面这种情况/ 先于! 出现:

```
#set( $foo = "bar" )
$/!foo
$/!{foo}
$//!foo
$///!foo
```

这样将被加工成

```
$!foo
$!{foo}
$/!foo
$//!foo
```

对比常规转义，/先于 $:

```
/$foo
/$!foo
/$!{foo}
//$!{foo}
```

这是结果是：

```
/$foo
/$!foo
/$!{foo}
/bar
```

15.4. Velocimacro 杂记

本节是关于Velocimacros的一个小型FAQ。本届内容会不时更新，所以请常来检查新的内容，

注：本节中，'Velocimacro' 将简写为'VM'。

Q:是否可以使用指令directive 或者 VM 作为另一个VM的参数? 例如: #center(#bold("hello"))

A: 不行。指令不能用作指令的参数，而大多数情况下，作为实际的应用，VM就是指令。

不过也有一些办法。一个简单的做法是使用双引号来加工你的内容。所以，你可以这样：

```
#set($stuff = "#bold('hello')" )
#center( $stuff )
```

甚至可以节省一个步骤：

```
#center( "#bold( 'hello' )" )
```

请注意，后面这个例子中，参数是在VM内部被求值，不是在调用的那一层次上。换句话说，被传入的VM的参数是整个被传入的，并且在传入的VM内部被求值。所以我们可以这样做：

```
#macro( inner $foo )
 inner : $foo
#end

#macro( outer $foo )
   #set($bar = "outerlala")
   outer : $foo
#end

#set($bar = 'calltimelala')
#outer( "#inner($bar)" )
```

这里，输入将会是：

```
Outer : inner : outerlala
```

因为"#inner($bar)" 的求值发生在#outer()内部，所以在#outer() 内设置的$bar得值会是其使用的值。

这是一个有意的保护特征—参数按名称传递给VM，所以可以将象状态引用的东西传给VM，比如：

```
#macro( foo $color )
 <tr bgcolor=$color><td>Hi</td></tr>
 <tr bgcolor=$color><td>There</td></tr>
#end

#foo( $bar.rowColor() )
```

rowColor() 被重复调用而不是一次。为避免如此，可以调用VM外部的方法，然后将值传递给VM.

```
#set($color = $bar.rowColor())
#foo( $color )
```

Q:是否可以通过#parse()注册VM？

A：当前， Velocimacros 在其首次在模版中使用前必须首先定义它。这意味着， #macro() 宣称应该在使用Velocimacros之前。

如果你想#parse() 一个包含#macro() 指令的模板，记住这个非常重要。因为#parse() 在运行时发生，解析器在解析时要决定是否模版中一个看起来像VM的元素真是VM，所以解析一系列VM 宣称可能并不能如愿地工作的很好。为避免如此，可以简单地使用velocimacro.library 的办法，使Velocity 在启动时载入VM。

Q. 什么是VM自动载入（Velocimacro Autoreloading）？

A. 这是一个属性，在开发时使用，而不时运行时：

```
 velocimacro.library.autoreload
```

默认值为false。当设置为true时，连同<type>.resource.loader.cache 属性设置为false（这里是使用的资源载入器的名称，比如'file')，Velocity 引擎在你创建VM库文件是将自动载入其改变，这样你就不必将其导入servlet 引擎（或者应用程序）中，或者用其他手段来使其自动重新载入。

下面是一个简单的设置配置组合：

```
file.resource.loader.path = templates
    file.resource.loader.cache = false
    velocimacro.library.autoreload = true
```

注意在生产状态（运行时）不要使其打开。

15.5. 字符串联

开发者常问的一个问题是“我如何进行字符串串联？”是否有类似于JAVA中的'+'操作符？

为了串联VTL中的引用，你不得不将它们“放在一起”。而你想要放置在一起的上下文很重要，下面举例说明。

在常规“笨办法”模板中：

```
         #set( $size = "Big" )
         #set( $name = "Ben" )

        The clock is $size$name.
```

输出将会是：'The clock is BigBen’。我们来看更有趣的事情，比如，当你想串联一个字符串并传递给一个方法，或者设置一个新的引用，可以这样：

```
        #set( $size = "Big" )
        #set( $name = "Ben" )

        #set($clock = "$size$name" )

        The clock is $clock.
```

结果是一样的。作为最后一个例子，当你想混合“静态”字符串到引用中，你可能需要使用“形式引用”：

```
  #set( $size = "Big" )
  #set( $name = "Ben" )
  #set($clock = "${size}Tall$name" )
```

现在，输出将会是'The clock is BigTallBen'。